

THE MEMORY WARP TUTORIAL

BASIC
BAY

POKE
CITY

COLOR
ISLAND

TRICKY TUTORIAL #11

PEEKSVILLE

BAIT

SAVE
LANDING

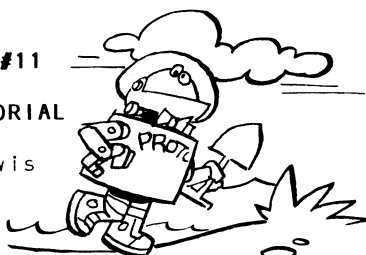
EDUCATIONAL
SOFTWARE, inc.

FOR ATARI® COMPUTERS



TRICKY TUTORIAL #11 THE MEMORY MAP TUTORIAL

by Robert L. Lewis



How to Load

TAPE....

With the BASIC cartridge in place, turn the computer on. Place the tape in your recorder, label side up. Make sure the tape is rewound, and reset the counter to zero. Push PLAY on the recorder, type CLOAD and press RETURN twice. If the program won't start to load, try positioning the tape forward or backwards a little. When the READY prompt appears, type RUN and press RETURN. We recommend you write down the number on your recorder's counter as each program starts, this will make it easier to find each part later on.

DISK....

To LOAD and RUN the disk, first turn on your disk drive. When the busy light goes out, place the disk in the drive. Now turn on the computer, with the BASIC Cartridge in place the program will LOAD each part and RUN by itself.

Requirements:

16K of memory
ATARI BASIC Cartridge

Optional:

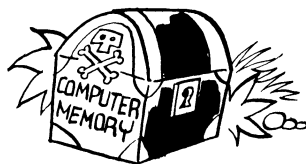
Joysticks and Paddles

Return any defective tapes or disks to:

Educational Software
4565 Cherryvale Ave.
Soquel, Ca. 95073

ATARI is a registered trademark of ATARI INC.

TABLE OF CONTENTS



HOW TO LOAD	1
INTRODUCTION	3
TABLE OF PEEKS AND POKES	27

#	Mnemonic	LOCATION	PAGE
---	-----	-----	----
1	POKMSK	16	8
2	RTCLOCK	18-20	9
3	SOUNDER	65	9
4	ATTRACT	77	10
5	LMARGIN	82	10
6	RMARGIN	83	10
7	ROWCRS	84	11
8	COLCRS	85-86	12
9	SAVMSC	88-89	13
10	OLDCHR	93	13
11	VNTP	130-131	14
12	STOPLN	186-187	15
13	ERRSAVE	195	15
14	PTABW	201	16
15	SDMCTL	559	17
16	SDLSTL	560-561	17
17	PADDLO-7	624-631	18
18	STICK0-3	632-635	18
19	PTRIG0-7	636-643	18
20	STRIG0-3	644-647	18
21	TXTR0W	656	19
22	TXTCOL	657	19
23	TABMAP	675-689	20
24	INVFLG	694	21
25	SHFLOK	702	21
26	CRSINH	752	22
27	KEYDEL	753	22
28	CHACT	755	23
29	CH	764	23
30	DSPFLG	766	24
31	CONSOL	53279	25
32	RANDOM	53770	25
33	PACTL	54018	26
34	COLDST	580	26

INTRODUCTION

A Memory Map is a list of key memory locations that the Operating System and BASIC initializes for various system functions. This Tutorial is intended for those of you who are using BASIC and have a desire to investigate and use the additional programming power of your ATARI computer. To point you toward this goal we will demonstrate PEEKs and POKEs in an interactive manner, especially designed so that you may gain experience by getting involved. Most of the memory locations that will be used to PEEK and POKE are maintained by your operating system and are valid locations for use with other languages. What you learn here can be applied to Assembly Language, Forth, etc.

SPECIAL HINTS FOR TAPE USERS

Six separate loads are required to see the entire tutorial. The lessons are not protected from accidental depression of the BREAK key. If the BREAK key is pressed by accident or to view some code be sure and press CTRL and CLEAR to erase the screen and then type RUN to go on. The above is needed to prevent errors from occurring due to files left open and variables dimensioned. Typing RUN automatically closes open files and clears variables. You would then need to advance to the lesson you need by pressing the ESC key.

As soon as you get a successful load, save each program on another back up cassette. This is a good habit to develop when using cassettes since data recorded on your recorder is likely to be played back by your recorder with greater reliability due to variations from one recorder to another.

THE SCREEN COLOR

Black and white users may want to modify the color selections to optimize the display for their sets. This can be accomplished by following the steps below.

- 1) Make a lesson selection from the Menu.
- 2) After lesson is LOADED type LIST and then press RETURN.
- 3) Press the BREAK key when the screen is half full.
- 4) Locate the third line number and find POKE 710,.
- 5) Replace the number following POKE 710, with POKE 710,0.
- 6) ReSAVE the modified lesson.

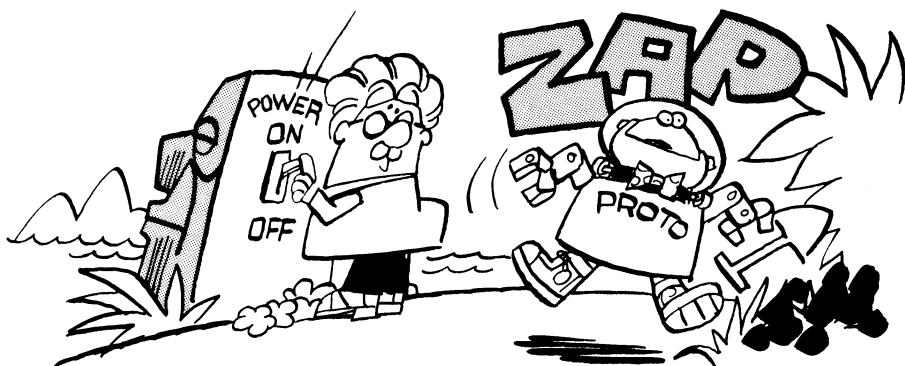
The above steps will produce a black and white display.

I AM HESITANT

Don't be! Your experimentation with PEEKs and POKES can be fun and much is to be gained in programming knowledge. The worst that can happen when you are experimenting with PEEKs and POKES will be your computer goes off into never-never land and won't respond to your command. There are solutions to this problem:

- 1) Press SYSTEM RESET, and if the computer still won't wake-up, then
- 2) Turn the power switch off, and back on, to bring your computer back to normal.

When you power up the computer, which is known as a cold start, the original values also called default values will be reset by the operating system. So, you see, it is impossible for you to hurt the computer by trying out ideas. You stand to learn a great deal, and if the computer refuses to respond just turn off and on the power switch and all will be well again.



WHAT'S A BIT AND A BYTE?

Inside your Atari computer are memory cells. Each cell holds one BYTE. A BYTE consists of eight bits. A bit is the smallest amount of information computers deal with. A Bit is seen as a (1 or 0), (on or off), (high or low). A BYTE consists of eight 0's or 1's in any data sequence ranging from all zeros, (00000000), to all ones, (11111111). All 0's in a byte equal decimal 0, and all 1's equal decimal 255. A byte can only hold a decimal value between 0 and 255. Any attempt to go outside that range will result in failure. The computer is not designed to accomodate larger numbers in one byte.

To figure out the decimal value a byte represents, you will need to design a chart and assign place holders to each 1 or 0 for the eight bits, see the second screen display for LESSON TABMAP for an example.

WHAT'S A PEEK?

No, you won't get arrested for being a PEEKING Tom! Learning to be proficient at PEEKing is easy and fun. Remember that each memory cell holds one byte consisting of eight bits. PEEK in BASIC allows you to look into any memory cell and see what is inside. In doing so its contents are not disturbed. A one-line BASIC program will return the binary (eight bit value) converted to decimal for us to see. The designers of ATARI BASIC have really made it as easy as possible for us. Each one of these memory cells has an address just like a house on a street. We can PEEK at any address in the memory range of 0 to 65536 using the following format:

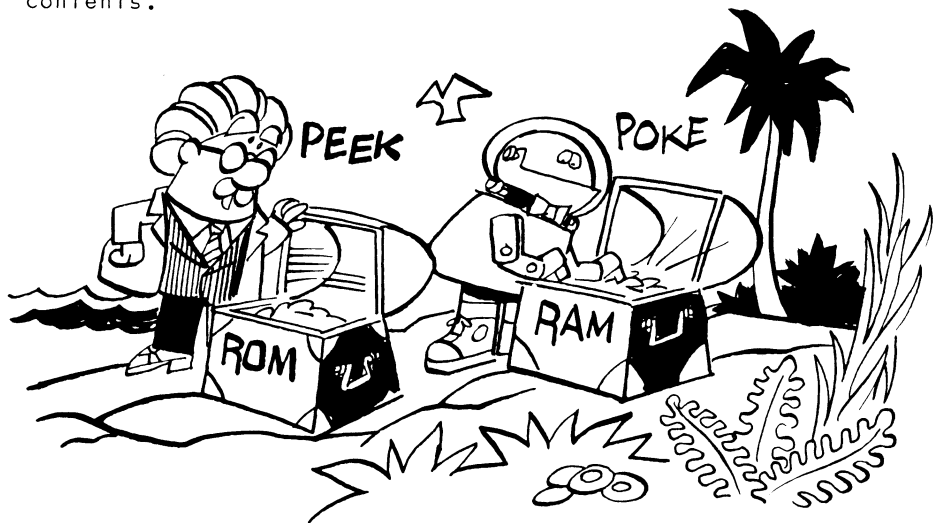
PEEK(address)

You may PEEK from direct mode and see the contents of any cell by using the form: PRINT PEEK(address-number). Where the number falls within the range of 0 to 65536. If you try a number outside of this range, the computer will give you an ERROR 3, A value error.

There are two types of memory cells inside your Atari:

1. RAM (RANDOM ACCESS MEMORY)
2. ROM (READ ONLY MEMORY)

Their functions differ. A ROM allows you to look, (PEEK), at it's memory locations, but you cannot place (POKE) any values into it. Any attempt to POKE into a ROM location will bring no results. The RAM memory allows you to look, (PEEK), at it's memory location, and to modify, (POKE), it's contents.



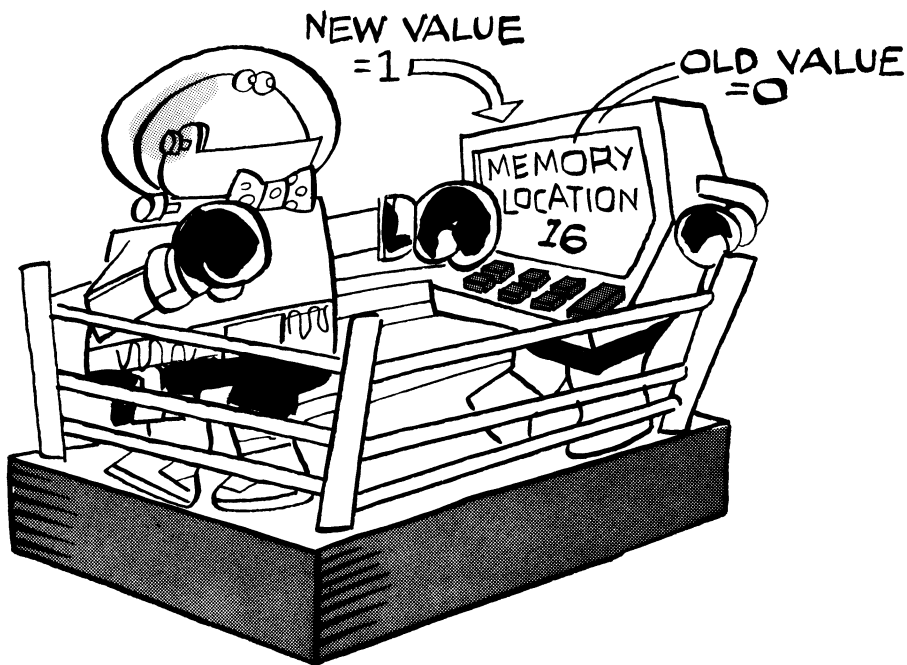
WHAT'S A POKE?

The POKE command in BASIC allows us to change the value in a memory cell to the value of our choice, i.e., throws out the old value and replaces it with a new one. We can only POKE values in the range of 0 to 255. We POKE using the following format:

POKE address,number.

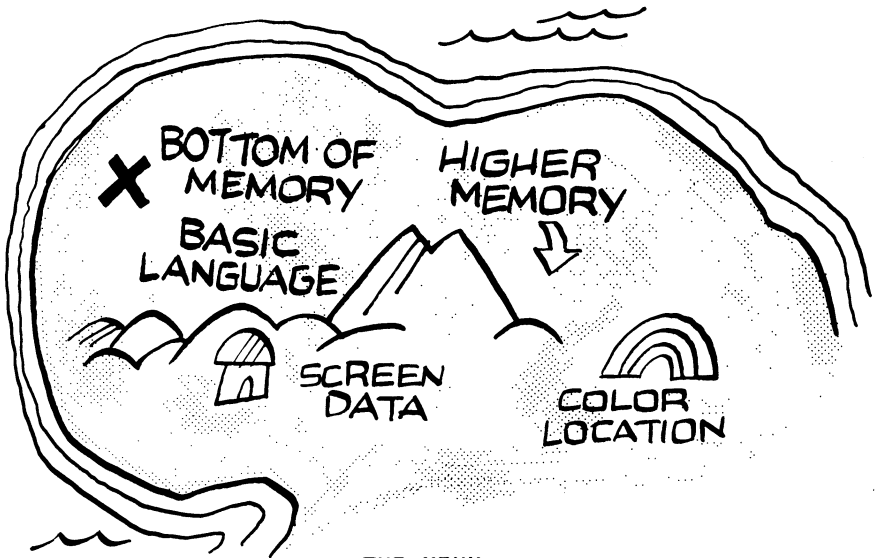
where address is a number between 0 and 65536. The second number is between 0 and 255. You cannot 'upset' the computer doing PEEKs, but using POKE to change values indiscriminately can cause the computer to get 'upset' thus requiring a cold start to get going again. To see the results of changing the contents of a memory cell try this simple program:

```
10 PRINT "WHAT NUMBER WOULD YOU LIKE TO POKE"  
20 PRINT "INTO MEMORY CELL 1536";  
30 INPUT X:POKE 1536,X  
40 PRINT "I SEE IN MEMORY CELL 1536 A ";PEEK(1536):GOTO 10
```



DIFFERENT POKES FOR DIFFERENT FOLKS!

One example of the use of POKES is to store data of your choice in memory cells. Later you may PEEK at these locations to recover your data. This tutorial will teach you some of the special features of this computer, so that you can learn locations that BASIC and the OPERATING SYSTEM use. You may tailor the POKE and PEEK functions to your own needs by studying this tutorial and having fun with MEMORY MAP locations.

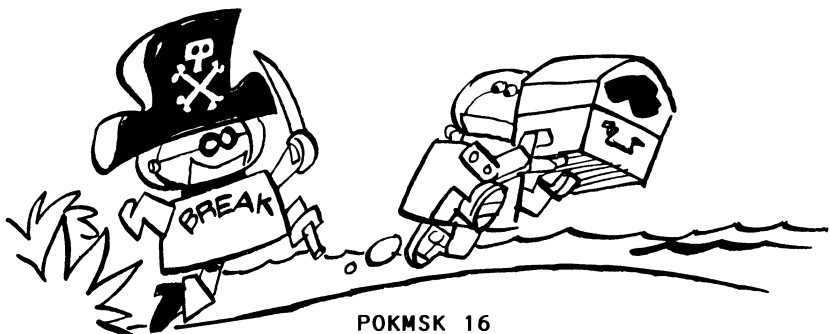


THE MENU

If you purchased the cassette version of this Tutorial skip over this part.

DISK users have been provided a menu that lists all the screen displays and their locations as covered by this Tutorial. If you wish to start with the first Tutorial Display press 1 or just press RETURN to find POKMSK 16, this is the first location discussed in the manual. From this first display location the following displays will then chain through each lesson. The other choice the Menu provides, is the means to select any lesson of interest. This latter method could be used when a reference to a location in an article is seen or when analyzing some code, and you wish to see if that location was included in this Tutorial. Each lesson allows you to return to the MENU and make other choices.

The MENU has several locations that are listed separately but use the same screen demonstration because the topics are closely related. They are listed separately so that their lesson can be easily found.



POKMSK 16

Those of you wishing to protect your programs from the undesirable results of the BREAK key being accidentally pressed and thereby having the program come to a halt can use POKE 16,64 and POKE 53774,64. The value 64 POKEd into locations 16 and 53774 will disable the BREAK key. Unfortunately, the changing of GRAPHICS modes, or the opening of an I.O.C.B. (INPUT OUTPUT CONTROL BLOCK) may reset these locations to their normal, default values. One way to overcome this short-coming is to include a subroutine in your program everytime you open a channel or change GRAPHICS modes you renew the protect feature, for example:

```
GOSUB 3000
30000 X=PEEK(16):IF X>128 THEN POKE 16,X-128:
      POKE 53774,X-128
30010 RETURN
```

This location is also known as the POKEY interrupt vector. When the BREAK key is pressed, an interrupt takes place, and the processor stops and services the interrupt. A routine separate from your program takes over and places the message "STOPPED AT LINE [so and so]" on the screen. You will also see the READY prompt allowing you to take over. Your program is still intact and you may type RUN or CONT or GOTO-line number. However, you may lose data, or your screen format may be gone. If you press BREAK be sure and press SHIFT & CLEAR to clear the screen, and then type RUN or LIST to avoid messing up the display.

Location 53774 is a Hardware Location which is shadowed in (RAM) at location 64. Shadowing is a process where values are moved between hardware locations and RAM locations, thereby allowing the program to monitor the contents of write-only hardware registers or check the inputs from read-only hardware registers.

There are other values that can be placed in this location. To learn about them, see the MASTER MEMORY MAP from Educational Software and DE RE ATARI from ATARI Inc.

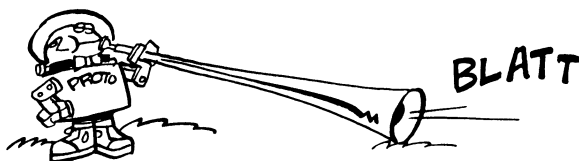
RTCLOCK 18-19-20

These three locations 18,19,20 store the Internal Clock. Every 1/60 of a second, location 20 increments by one until 255 is reached, then location 20 resets to 0 and location 19 increments by 1. Every time location 20 finishes a complete cycle (0-255), then location 19 will increment by 1. If location 19 started out at 0, then it would take location 20 going through its cycle 255 times for location 19 to increment to 255. When location 19 cycles back to 0, then location 18 will increment by 1. It takes 1/60 times 255 (4.25 seconds) for location 20 to complete a cycle. Location 19 then takes 4.25 times 255=1083.75 seconds or 18.0625 minutes. Location 18 then takes 18 minutes times 255=4605.9375 minutes or 76.765625 hours.

As you can see, either small periods of time, called jiffies (1/60 second), or long periods of time can be measured. You should note that these locations are maintained by the Operating System independent of BASIC. They go on counting by themselves whether BASIC is running or not. You may POKE any number into these locations between 0 and 255 in order to set them to your needs and then PEEK at them occasionally in your program to see if the proper time has elapsed.

We encourage you to dream up uses for these locations and experiment with them. Try placing a 254 in location 19 and a 255 in location 18, and then watch all three locations recycle back to zero. The following short program will demonstrate one possible use of these locations:

```
10 POKE 18,0:POKE 19,0:POKE 20,0
20 IF PEEK(19)=3 THEN GOTO 40
30 GOTO 20
40 SOUND 0,65,10,8:IF PEEK(19)=4 THEN END
```



SOUNDER 65

Location 65 normally has a 3 in it, and allows the user to hear the sound of data being transferred during the input/output operation to the Disk or Tape. Some folks like to hear the sounds of data transfer, since the sound is unique for each process, and can warn if a problem is occurring. If location 65 is POKEd with a zero(0) then the beep-beep-beep sounds heard during data transfer will not be heard. This may be preferred by those who find the sound annoying.

ATRACT 77

A permanent image can be burned into the phosphor of your television screen if an unchanging pattern is left on for many hours. Picture tube manufacturers may void your warranty if they see this burn-in. The Atari engineers have built into the computer a system that will vary the screen pattern when 9.01 minutes have elapsed without keyboard activity. If you do not want to wait 9.01 minutes you can activate Atract immediately by poking location 77 with a 254. Location 77 is reset to normal, 0, anytime any key is depressed.

Without keyboard activity, this location increments until 128 is reached and then luminescence is reduced and colors rotate to protect the television screen. Games that use joysticks need to have code included that resets this counter during joystick movement or the Atract mode will be activated.

```
10 IF PEEK(632)<>15 THEN POKE 77,0
```

The above code will reset 77 to 0 if joystick 1 is moved.

LMARGIN 82 / RMRGIN 83

Many T.V. receivers do not have the width control adjusted correctly therefore the picture display extends beyond the right and left edges of the picture frame. In T.V. entertainment viewing this short-coming may not be critical, but for computer viewing it cannot be tolerated. Therefore, to overcome the possible loss of sight of data on the edges, the left margin is preset to 2 instead of 0. The right margin is preset to 39. The above margin values, 2 and 39, are stored in locations 82 and 83 when the computer is turned on. You may look at these locations and verify their contents using the following short program:

```
PRINT "THE LEFT MARGIN=";PEEK(82);  
" THE RIGHT MARGIN=";PEEK(83)
```

You can change the preset values to suit your requirements with the following POKE program:

```
POKE 82,X or POKE 83,Y
```

where X and Y are your choices for the left and right margin dimensions. Seeing the Screen Display section of this lesson will inspire you to use the MARGIN locations for many programming design needs.

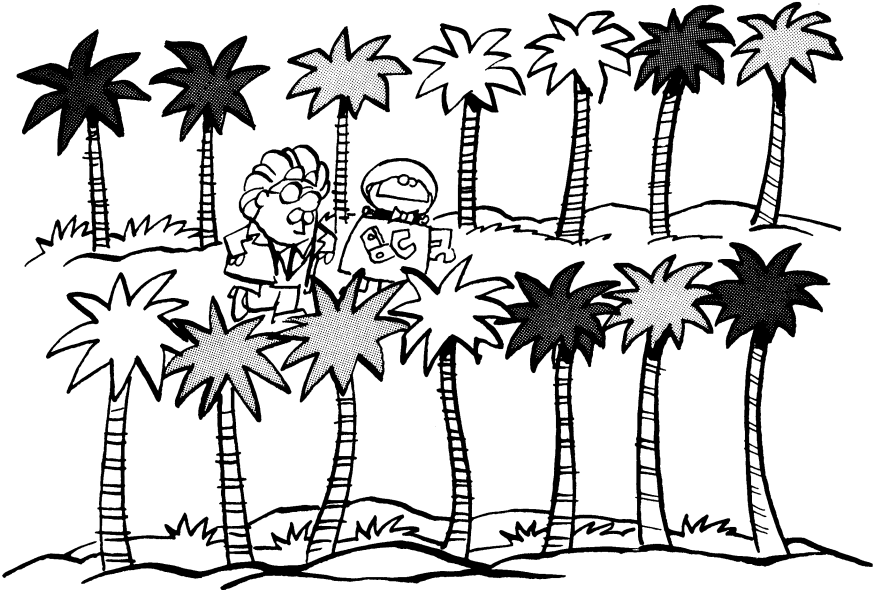
ROWCRS 84

GRAPHICS Mode 0 assigns 24 rows and 40 columns for the dimensions of the display screen counting from zero. A row is defined as extending from the left margin to the right margin of the screen and is called the x-axis of the display. The rows are identified by number; the first row, starting at the top left, is 0, and the last row, at the bottom, is row 23. You can verify the number of rows by locating the cursor at the top of the screen and actuating the RETURN key, counting the number of actuations, until the cursor reaches the bottom of the screen.

A programmer may select any row between 0 and 23 to point the cursor where the next print will appear. This is done with the use of a POKE program, for example, the following program will place an X at the 10th row.

```
POKE 84,10:PRINT "X";
```

See the SCREEN DISPLAY section of this lesson for a demonstration using the location ROWCRS. It will stimulate your program application imagination.



COLCRS 85/86

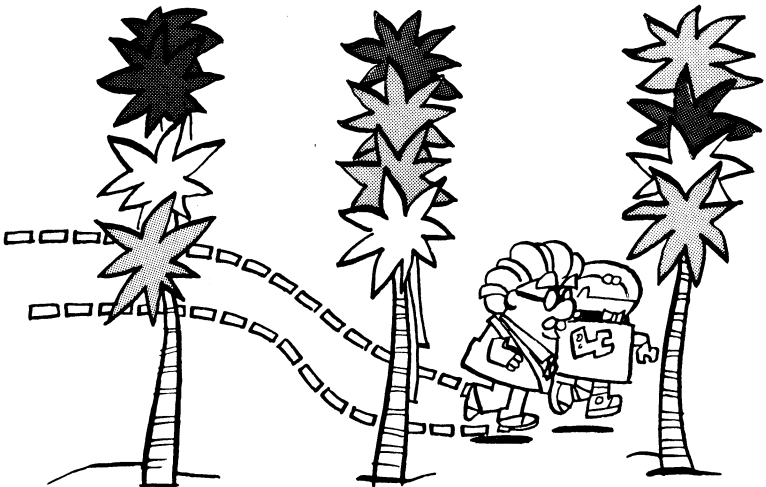
GRAPHICS Mode 0 assigns 24 rows and 40 columns for the dimensions of the display screen. A column is defined as extending from the top to the bottom of the screen and is called the y-axis of the display. The columns are identified by number; the first column, starting at the top left corner of the screen, is column 0, and the last column, at the right edge of the screen, is column 39. You can verify the number of columns by locating the cursor at the top of the screen and actuating the SPACE BAR key, counting the number of actuations, until the cursor reaches the right edge of the screen.

A programmer may select any column between 0 and 39 (40 columns) to point the cursor where the next print will appear. This is done with the use of a POKE program. For example, the following program will place X at the 10th column.

```
POKE 85,10:PRINT "X";
```

In this lesson, only location 85 is used even though location 86 is displayed. Location 85 can point to any column between 0 and 255. This is more than enough for graphics mode 0. The only time location 86 would be used would be for graphics modes 8 through 11, which have more than 255 columns.

See the SCREEN DISPLAY section of this lesson for a demonstration using the location COLCRS.

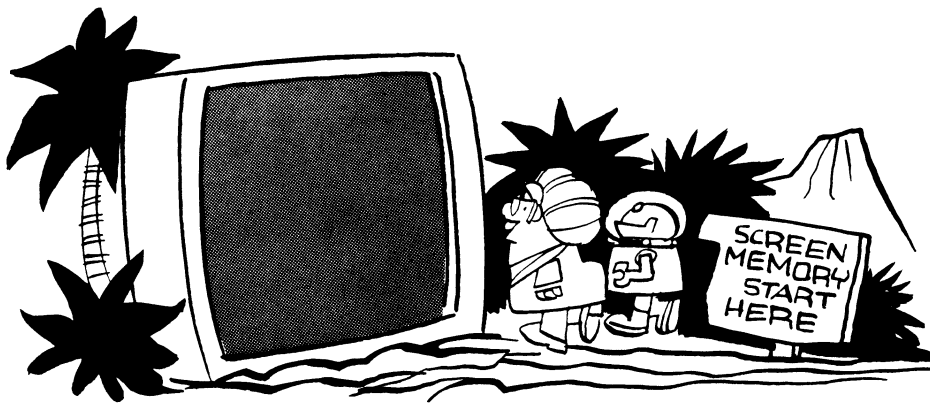


SAVMSC 88-89

This location determines the address in memory where the information displayed on the screen is stored. This data will begin displaying in the upper left hand corner of the screen. Each successive memory location will print along that row until it wraps around to the start of the next row. The formula to determine where the screen memory begins is:

$$\text{ADDRESS}=\text{PEEK}(88)+\text{PEEK}(89)*256$$

The ATARI computer uses a procedure called memory mapping. The exact location of memory that is used to support what is on the screen varies with the Graphics mode and the amount of RAM in place. In the back of your BASIC Reference Manual is a TABLE OF MODES AND SCREEN FORMATS that defines the amount of RAM required for each GRAPHICS mode. Using the above formula after a called GRAPHICS mode and adding the total RAM from the chart will then tell you where the end of screen memory is. Using this location gives you absolute control over what appears where on the screen. You also can overcome scrolling problems due to the BASIC editor during screen print.



OLDCHR 93

This location holds the value of a character under the cursor. It is used by the operating system to restore a character to normal when the cursor passes by. You can also position the cursor to a location on the screen and read the value under the cursor. I bet you didn't know a cursor could read! Well, give it a PEEK TEST. I encourage you to allow your imagination to run rampant and dream up uses for some of the locations talked about but, no amount of talking and demonstrating can equal your imagination and effort trying out these locations in your own work.

VNTP 130-131

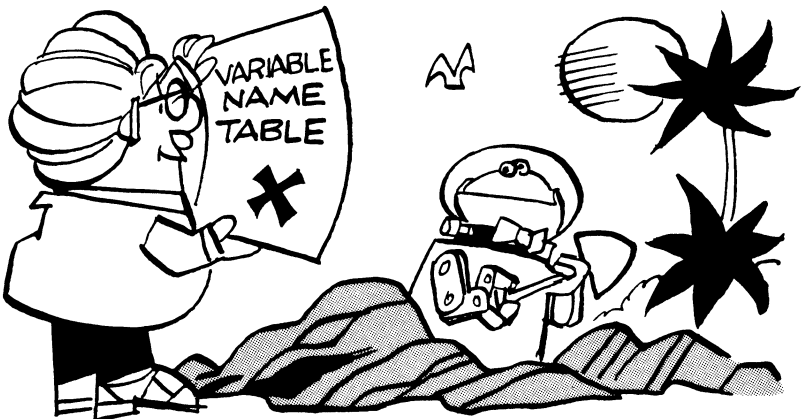
These locations hold the starting address of the variable name table. We may see the names of all variables entered when the program was RUN. Each variable is stored in ATASCII in the order entered. The last character is in reverse video as an end flag.

If you ran a program and then changed your variable names, or deleted some, the deleted names still reside here. Since a maximum of 128 variable names can be defined by this BASIC, you may find it advantageous to check this list for variables no longer in use. If you find variables not used by the program, get rid of them by LISTing your program to DISK or TAPE. Type NEW, and then ENTER them back into memory.

The software for the DISPLAY section of this lesson uses VNTP, location 132-133, which defines the end of the variable name table. BASIC uses this pointer to find the end of the variable name table. When there are less than 128 variables this normally points to a dummy zero byte. When 128 variables are present, this points to the last byte of the last variable name.

If you would like to look at the list of variables you have used in your program try the following program:

```
10 J=PEEK(132)+PEEK(133)*256
20 PRINT J;:REM END OF VARIABLE NAME TABLE
30 K=PEEK(130)+PEEK(131)*256
40 PRINT K;:REM START OF VARIABLE NAME TABLE
50 FOR L=K TO J
60 PRINT CHR$(PEEK(L));" ";:NEXT L
```



STOPLN 186-187 / ERRSAVE 195

If you use the TRAP statement in your programs to prevent an ERROR message from halting your program, then you will find that PEEKING at locations STOPLN and ERRSAVE will be very useful. These locations are only PEEKED at, they are not POKED.

Location 186-187 will hold the program line number where a STOP or TRAP occurred. It may be looked at, and its contents seen by using:

```
PRINT PEEK(186)+PEEK(187)*256
```

Location 195 will hold the ERROR # that caused the stop. You can look at and see the contents of this location by using the following:

```
PRINT PEEK(195)
```

Using the above locations in a subroutine in conjunction with the TRAP statement can give the programmer information without the program stopping. A useful program using this technique, giving the programmer full, customized textual clues to an error, can be found in the Nov.-Dec. 1980 issue of Compute! Magazine. Pg. 58.

The DISPLAY section of this lesson is an example of what can be done with the PEEKING of these locations.

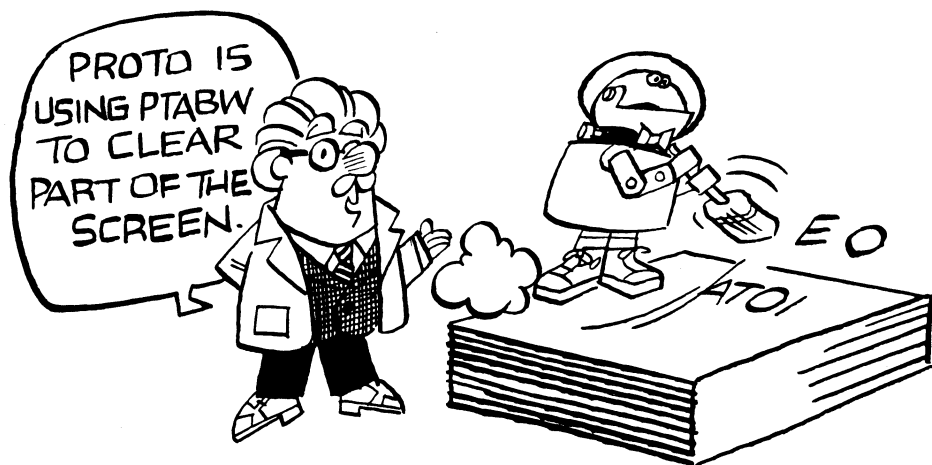


PTABW 201

This location has nothing to do with the TAB key. It does control how far over the cursor will tab when a comma is encountered as a print separator. When the computer is first turned on, the operating system loads a 10 into this location. If a comma is encountered while the cursor is within the first 10 columns, then the cursor will only jump to the 10th column. With the cursor at the default value of 2, a comma would cause a jump to the 10th column (8 spaces) and another comma would cause the cursor to jump to column 20 (10 spaces). Fortunately, the wizards at ATARI have given us a location whose value represents how many spaces a comma represents. Change this value to suit your programming needs.

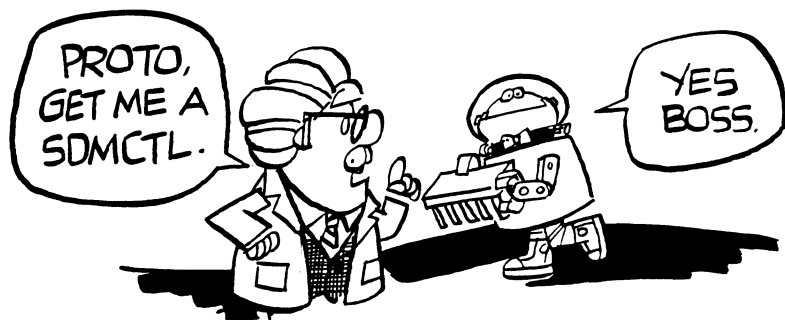
NOTE: A comma represents multiple spaces so remember that it can be used to clear parts of the screen. For example, `POKE 201,40:PRINT ,,,` would clear 160 spaces on the screen from your selected print position.

The comma acts as if you pressed the space bar the number of times found in location 201. The action you see in the display lesson, of your last name moving, is caused by the repeated printing of your first name and last name. The comma erases the old printed last name by spacing over the last name and then reprints the new last name one position to the right.



SDMCTL 559

This location controls a chip inside your ATARI computer called ANTIC. Its job is to control Direct Memory Access (DMA), which is the screen display chip. It uses a process called cycle stealing from the 6502 Central Processing Unit (CPU). The Antic redraws the TV screen 60 times per second. All this cycle stealing takes time away from the processor and slows it down by 30% or more. By turning Antic off (POKE 559,0) you allow the CPU to devote all of its time to processing. The result of this POKE is that the screen will go blank since the Antic has been turned off. When the work has been completed by the CPU, remember to turn Antic back on by a POKE 559,34 or your results won't be visible. Most of the DISPLAY lessons use this technique so that the writing you see on the screen appears all at once. Other values and uses of location 559 are covered in Tricky Tutorial #5, Player/Missile Graphics.



SDLSTL 560-561

This is the pointer for the starting address of the display list. The list is a special set of instructions the Antic Chip obediently follows to display the various Graphics Modes your program calls out. A customized screen can be displayed by modifying the display list giving you many possibilities for mixed graphics and text modes not available from BASIC's predetermined GRAPHICS 0 to GRAPHICS 11 Modes. In these lessons, you are observing a modified display list. Feel free to use the code from any of these lessons to give your programs a large title line at the top of the screen. This display is not from one of the normal BASIC Graphics modes. A complete discourse on Display Lists is not the purpose of this lesson. We only show you how to find the start of the display list. The formula for the start of the Display List is:

$$DL=PEEK(560)+PEEK(561)*256$$

A complete discussion on Display List modification can be found in Tricky Tutorial #1.

PADDL0-7 624-631

and

PTRIG0-7 636-643

This lesson demonstrates that memory locations are used to determine the status of all Paddles and triggers. The Paddle ports are monitored by the Operating System even when your program isn't asking for a value. The Paddles are potentiometers (variable resistors). The resistance is digitized and displayed as a value in each assigned memory location. Since we can have 8 Paddles operational at one time, it takes 8 memory locations to track them. We also have 8 memory locations to track the triggers. See the Display lesson, and test all your paddles and triggers to see if they operate properly. Observe the values returned by each memory location for the triggers and paddle rotation.



STICK0-3 632-635

and

STRIG0-3 644-647

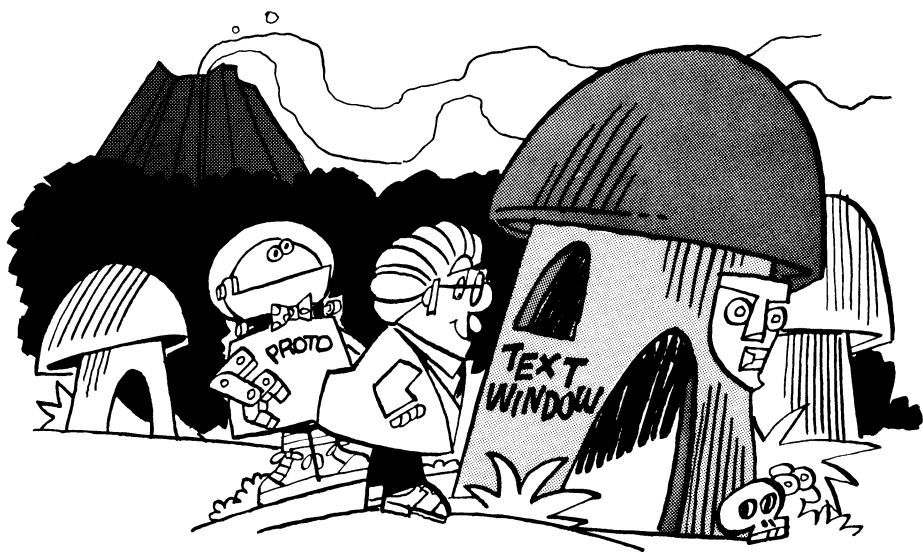
This lesson demonstrates that memory locations are used to determine the status of all Joysticks and Triggers. The Joystick ports are monitored by the Operating System even when your program isn't asking for a value. The Joysticks are switches (on or off). There are four switches for Joystick direction. The diagonal movement is sensed by two switches showing a path to ground. The fire button also has a path to ground when it is pressed. For every direction the Stick is moved, a unique value is returned. This value is stored in an assigned memory location for that Stick. Since we can have 4 Joysticks operational at one time, it takes 4 memory locations to track them. We also have 4 memory locations to track the triggers. See the Display lesson, and test all your Joysticks and Triggers to see if they operate properly. Observe the values returned by each memory location for the Triggers and Joystick direction.

TXTR0W 656 AND TXTCOL 657

The POSITION command in BASIC doesn't cover the text window. Fortunately ATARI has provided us with some POKE's that allow us to determine where in the text window the print-out will appear. Location 656 determines the ROW where the next print will appear. We POKE 0-3 into 656 to print text to a selected ROW. Remember the text window is seen at the bottom of the screen after you type GRAPHICS 1 thru GRAPHICS 8. The text window only has 4 possible print rows so if you desire your print to appear in row 2 just POKE 656,2 and then print your message.

Location 657 can be loaded with the column number you wish your next printed output to appear. A number between 0 and 39 would be chosen, since the text window is limited to a 40 column screen. To print the word ATARI in the third ROW and the twentieth COLUMN you would:

```
POKE 656,3:POKE 657,20:PRINT "ATARI"
```



TABMAP 675-689

Memory locations 675 through 689 have been set aside to store Tab stops. Any of 120 possible TAB stops can be stored by POKEing the above locations with numbers representing the desired tab stop positions. The graphics mode chosen determines the number of columns that can be set with stops. In graphics mode 0 we have 40 columns. Each location #, (BYTE) covers 8 columns. Location 675 covers the first 8 columns. Location 676 covers the next 8 columns. Location 677, the next 8, until all graphic mode requirements are met. Only 5 locations are required to support a graphics mode 0 screen, 8 bits times 5 bytes=40 tab locations.

Why then do we see 15 locations for this lesson? Because in higher graphics modes, we have more columns. NOTE: TABS are settable in GRAPHICS 8. Graphics 8 has 320 possible columns. Location 675-689 is 15 bytes, 8 bits times 15 bytes=120 possible TAB stops, this won't cover all 320 possible columns but gives the programmer enough flexibility to cover most any situation. Some knowledge of binary is required to use the TABMAP locations and it is suggested that you refer to the DISPLAY section (2nd.frame) of this lesson for a review of binary and decimal number conversion.

The left-most column, if set by a TAB, is the most significant bit and counts as decimal 128. The next column, if set by a TAB, counts as decimal 64. The next counts as 32,16,8,4,2,1 until all 8 columns, left to right, are accounted for. We add the decimal numbers for the first eight positions and POKE that number into the first location. This process is repeated with the second byte representing the next 8 columns until all required TAB stops are set. A short program to clear all TAB stops is seen below.

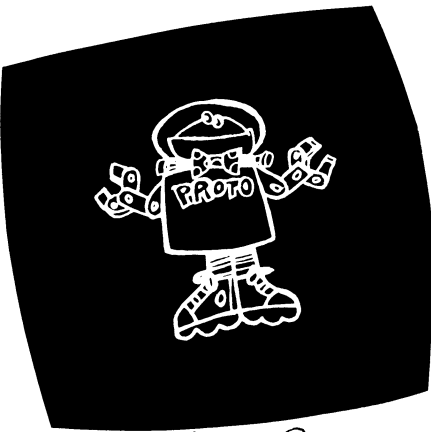
```
10 FOR I=675 TO 689:POKE I,0:NEXT I
```

High Order Bit							
↓							
Decimal value	128	64	32	16	8	4	2 1
Bit #	7	6	5	4	3	2	1 0
Binary contents	1	0	0	1	1	1	1

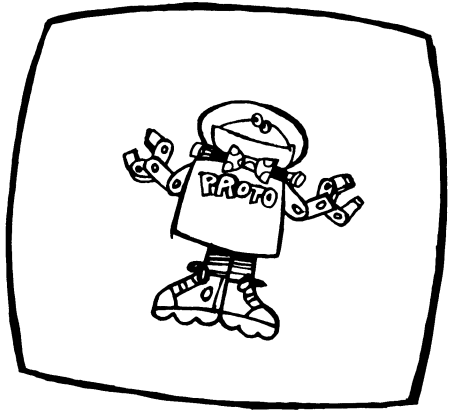
This diagram represents how the number 159 is stored in memory.

INVFLG 694

This location allows you to force a switch to reverse video and back normal again during INPUT from the keyboard. It acts the same as if the special ATARI key was pressed. If you desire INPUT from the keyboard to be in reverse video, then POKE 694 with a 128. To switch back to normal, POKE 694 with a 0. The special Atari key still works with this POKE and will switch between normal and reverse video. In the DISPLAY section of this lesson we switch back and forth with each typed character to demonstrate its speed and flexibility. In actual programming use, the switch would remain in place for longer duration in all probability. Try the INVERSE VIDEO FLAG in your programs.



694=128



694=0

SHFLOK 702

The choice of the numbers below, when POKEd into this location, allows the programmer to determine if the next input from the keyboard will be in:

- 0 =lower case letters only accepted.
- 64 =UPPER CASE LETTERS ONLY ACCEPTED.
- 128 =ATARI graphics characters only.
- 255 =Ignores all letters but will print everything else.

Try POKEing the above numbers into location 702 to observe if your results match the chart. Remember the effect takes place on the next print following your POKE.

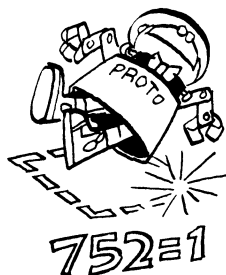
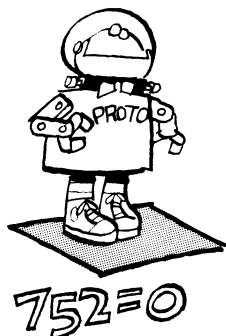
CRSINH 752

Did you know that it is possible to print without a cursor being on? Yes, the letters will still appear, but the cursor will be invisible. The invisible cursor can make it difficult to find where on the screen you will be typing. Leave it on when input from the keyboard is being requested. Turn it off to get rid of distracting cursor flashes when printing text or updating numbers as demonstrated in the DISPLAY section of this lesson.

POKE 752,1 makes cursor invisible.

POKE 752,0 makes cursor visible.

With one stroke of the keyboard you too can be a magician and make something invisible.



KEYDEL 753

All mechanical keyboards have a problem called key bounce. This problem comes from the keyboard contacts opening and closing as you type. The mechanical action, as seen in high speed photography, or by our very fast computers would show an instant where the key is bouncing between the on and off state until the contact finally seats and makes a good electrical contact. If this problem is not addressed by the computer designers, the results can be catastrophic to the user. For example, if you pressed p once several pppppp's might appear in a row, and as you can see, this means extra work, and frustration results. Many early model computers had this problem. A software fix was finally made available. The Atari solution to this problem is to load a 3 in location 753 every time a key is pressed. The 3 is then decremented every 1/60th of a second until 0 is reached. Until the 0 is reached, the keyboard isn't being scanned for input. This allows the contacts to seat and stabilize. This is called the keyboard debounce routine.

You may PEEK at this location, looking, for a value greater than 0 to see if a key has been pressed and if it has then go to some routine in your program.

CHACT 755

This location can be used to show off what one ATARI POKE can do. It would be difficult to try to duplicate the following amazing tricks on other computers, it could take 8000 bytes of memory to duplicate the techniques done here in 1 byte-- if it could be done at all on other computers (APPLE users, eat your heart out!).

POKE 755, with the number on the left below.

- 0 Makes all inverse video normal video.
- 1 Makes all inverse video invisible.
- 2 Default value (makes everything normal).
- 3 Makes all inverse video into solid blocks.
- 4 Same as 0 but upside down.
- 5 Same as 1 but upside down.
- 6 Same as 2 but upside down.
- 7 Same as 3 but upside down.

Why did ATARI designers choose to support upside down characters? Many arcade games use mirrors to reflect what's seen on the picture tube to the user. This allows the mirror to be painted with a background that is independent of the computer software. The upside down characters are fun to look at, but we'll let you dream up a practical use. When viewing the display lesson observe the arrow on the left pointing to the line being referenced.

CH 764

This location holds the Internal Code of the last key pressed. POKE a 255 here to clear the register. POKE a 12 here, and the RETURN key would not need to be used.

The term, "the last key", means if you pressed S and then followed with an A, only the code for A, the last key, would be stored in location 764. When you are PEEKing at this location, remember that Internal Code and Atascii Code are different. See your BASIC manual for more information.

This location allows you as a programmer to sense for a particular key being pressed without having to open an I.O.C.B. to the keyboard and do a GET key. It also allows you to PEEK for the YELLOW keys, located at the right of the keyboard, and a keyboard key at the same time.

All of the routines used in these lessons use location 764 so you may want to study the code by listing one of the programs.

DSPFLG 766

This location is used when you wish to print a control function symbol, but not perform it. Several of the the DISPLAY lessons use this POKE to display the cursor control arrows. The normal value in this location is a 0.

Use POKE 766,1 to display the character that represents:

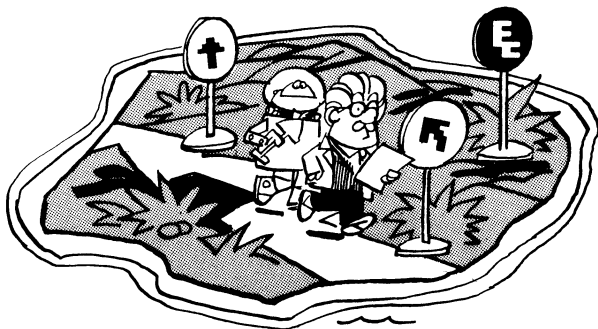
UP, DOWN, RIGHT, LEFT, CLEAR,
TAB SET, TAB CLEAR, LINE INSERT,
LINE DELETE, CHARACTER INSERT,
CHARACTER DELETE, TAB, and
DELETE BACK SPACE KEYS.

Remember while POKE 766,1 is in effect none of the above keys will be functional.

To print a reverse video right hand arrow, look in Appendix C (Atascii Character Set) of your BASIC Reference Manual. Find the decimal number for the right hand arrow which is 31, then add 128 to the 31 for 159. The 128 added to any Atascii number turns on the 8th bit of the binary code which the ATARI Computer recognizes as a flag to print in reverse video. See the diagram below which demonstrates this process. A sample program line for the above follows:

```
10 POKE 766,1:PRINT CHR$(159):POKE 766,0
```

Don't forget to POKE 766,0 when done printing or the special functions won't work when you need them.



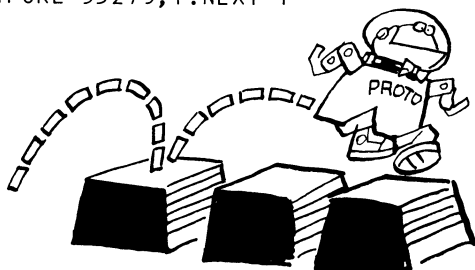
CONSOL 53279

The three yellow console switches on your right are only readable by PEEKs. These keys are used often in programs, and we encourage you to practice using these locations soon. Eight combinations are possible (0-7) as seen in the chart below. The number on the left represents the actual value returned by a PEEK at 53279.

- 0 All three yellow keys pressed simultaneously.
- 1 SELECT and OPTION keys pressed simultaneously.
- 2 START and OPTION keys pressed simultaneously.
- 3 OPTION key pressed.
- 4 START and SELECT keys pressed simultaneously.
- 5 SELECT key pressed.
- 6 START key pressed.
- 7 No keys pressed.

The second half of this lesson demonstrates a POKE that will cause the internal speaker to buzz. The buzz is raspy-sounding and can get one's attention. The following program will cause this buzz.

```
10 FOR I=1 TO 500:POKE 53279,1:NEXT I
```



RANDOM 53779

This location holds a random number generator which is independent of BASIC. By using the following format:

```
RANDOM=PEEK(53770)
```

a random number between 0 and 255 will be returned. You can multiply, add, or apply any math to the variable RANDOM. This allows you to define the range of numbers needed.

Compared to using BASIC's random number generator, i.e., `RANDOM=INT(RND(0))*256`, the POKE method seems easy and takes less typing.

PACTL 54018

This location is used to turn on and off the cassette motor. POKE 54018,52 to turn on the cassette motor and hear what is on the cassette through your television speaker. POKE 54018,60 will turn off the cassette motor.

If you have a 410 PROGRAM RECORDER, you may put an ordinary music cassette into your machine. Remember to push the play button. With the POKE on, music will be heard through your television speaker. This is great for listening to your favorite POP group, through your computer, while programming! Be careful not to do DISK I/O., as the cassette running may interfere with the data transfer. The use of this location is essential for cassette I/O called from program statements.



COLDST 580

If the SYSTEM RESET is accidentally pressed while you are working with disk-based programs it's possible to get back on line. POKE location 580 with a 1 and then press SYSTEM RESET, the equivalent of a cold start takes place--like turning your power switch off and on.

If a disk with an AUTORUN.SYS file, pointing to a MENU is in the drive, then the MENU will automatically appear. If you don't have an AUTORUN.SYS file on the DISK, then you will see the READY prompt after DOS reloads. The normal value in this location is 0.

Tape users will go to the READY prompt upon pressing SYSTEM RESET.

TABLE OF PEEKS AND POKES

To demonstrate just how useful PEEK and POKE memory locations are, the following table has been compiled. It shows the number of times each location was either PEEKed at or POKEd into during the course of writing this tutorial.

Mnemonic -----	LOCATION -----	# of PEEKS & POKES used -----
POKMSK	16	20
RTCLOCK	18-20	22
SOUNDER	65	14
ATTRACT	77	27
LMRGIN	82	28
RMRGIN	83	28
ROWCRS	84	26
COLCRS	85-86	33
SAVMSC	88-89	26
OLDCHR	93	23
VNTP	130-131	16
STOPLN	186-187	16
ERRSAVE	195	16
PTABW	201	17
SDMCTL	559	21
SDLSTL	560-561	21
PADDLO-7	624-631	53
STICK0-3	632-635	48
PTRIG0-7	636-643	53
STRIG0-3	644-647	48
TXTR0W	656	14
TXTCOL	657	14
TABMAP	675-689	63
INVFLG	694	25
SHFLOK	702	32
CRSINH	752	28
KEYDEL	753	14
CHACT	755	22
CH	764	16
DSPFLG	766	19
CONSOL	53279	31
RANDOM	53770	10
PACTL	54018	23
COLDST	580	9

NOTES



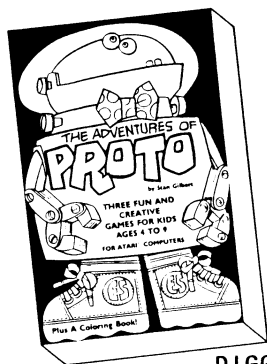
**MAKE LEARNING TO PROGRAM FUN
WITH
THE TRICKY TUTORIALS (tm)**

There are Tricky Tutorials to help you with everything from how to use your computer to programming your own special sound and graphic effects. The Professor and Proto un-complicate these functions and make learning to program fun!

- #1 - DISPLAY LISTS (Mixing many Graphics Modes on the screen at once.)
- #2 - SCROLLING (Move Text and Graphics up and down or across your screen smoothly.)
- #3 - PAGE FLIPPING (Program screen changes to switch instantly, like flipping pages in a book.)
- #4 - BASICS OF ANIMATION (A beginner's guide to creating animated shapes and simple games.)
- #5 - PLAYER MISSILE GRAPHICS (Learn to write arcade-style games.)
- #6 - SOUND & MUSIC (From single notes to songs and special effects, all explained clearly.)
- #7 - DISK UTILITIES (Seven utilities to help you use your disk drive all wrapped up in this one package.)
- #8 - CHARACTER GRAPHICS (You can edit the character set in your Atari for use in your own programs, even arcade-style games.)
- #9 - GTIA- GRAPHICS 9 to 11 (A veritable rainbow of nine colors and sixteen shades makes your screen come alive!)
- #10 - SOUND EFFECTS (Tweets, whistle, and roars from your Atari? Yes, it's all possible with this Tutorial.)
- #11 - THE MEMORY MAP TUTORIAL (Professor Von Chip explains his favorite tool to help you utilize your Atari's memory locations.)

...And look for new Tricky Tutorials coming soon to your favorite software store!

ALSO FROM EDUCATIONAL SOFTWARE

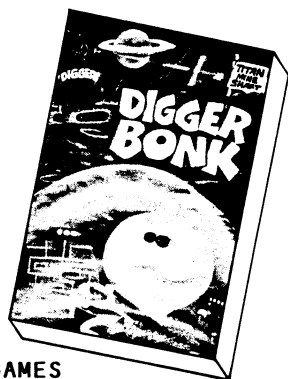


THE ADVENTURES OF PROTO

by Stan Gilbert. Here are three creative adventures to help the friendly robot/alien, Proto, learn about our planet. Includes MARS-MALLOWS, COLORING BOOK, PLAYTUNE as well as a storybook with pictures for coloring. 16K-Tape/32K-Disk.

DIGGERBONK

by Steve Robinson. An alien finds himself on the moon Titan where he must clear out a dangerous mineshaft. His faithful robot, Digger, ventures in to "bonk" the Purple Gurples and other deadly enemies contained in the continuously scrolling mineshaft maze that never repeats. 16K-Tape/32K-Disk.



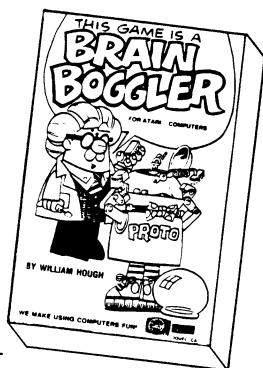
SPACE GAMES



by Randy Massey. Three exciting arcade-style games for one very affordable price. Includes ALIENS, an "Invaders" type game, SURVIVE, where you find yourself located in a very strange and dangerous universe, and ROBOT ATTACK, where you'll have to make your way through many rooms of aliens in an attempt to capture the cloaking device. 16K-Tape/32K-Disk.

BRAIN BOGLER

by William Hough. Music, sound, and color combine to add to the excitement of finding the secret code hidden deep within the computer's memory. This game is a real test of your powers of deductive logic. Will you find it before time runs out? 16K-Tape/32K-Disk.



IN THE UNLIKELY EVENT YOU
RECEIVE A DEFECTIVE PRODUCT
RETURN IT TO US FOR A
PROMPT REPLACEMENT OR REFUND

Let your computer help you unravel its inner mysteries. You may be familiar with our Master Memory Map, already a standard reference book for many ATARI® owners. Well, this tutorial goes beyond just reference! Professor Von Chip explores dozens of key memory locations and explains them with many sample programs. He clearly demonstrates such "tricks" as making the cursor invisible, variable margins and columns, creating flashing displays, programming games with joysticks or paddles and much more.

POKE & PEEK MENU			
#	NAME	LOCATION	# NAME LOCATION
1	POKMSK	18-20	18 PTRIGO-2636-643
2	RTCLCK	18-20	19 STRIGO-3644-647
3	SOUNDER	65	20 IXTRON
4	ATRACT	77	21 IXTCOL 657-658
5	LMRCIN	82	22 TOBMAC 675-689
6	RMRCIN	83	23 INVFLG 694
7	ROMCR9	84	24 SHFLCK 702
8	COLCR5	85-86	25 CR3INH 752
9	SOUNSC	86-89	26 KEYDEL 753
10	OLDCHR	93	27 CHACT 755
11	UNTP	130-131	28 CH 766
12	STDPN	186-187	29 DPFLE 766
13	PTABW	201	30 CON5OL 53-727-9
14	SDMCTL	559	31 RANDOM 53-770
15	SDLSTL	560-561	32 PDCLE 548-18
16	PADDL9	7624-631	33 COLDET 580
17	STICK8	3632-635	34 EXIT

Enter your choice (1-34)?■

REQUIRES:
16K TAPE / 32K DISK
BASIC Cartridge
JOYSTICK OR PADDLES OPTIONAL

EDUCATIONAL SOFTWARE, inc.
4565 Cherryvale Avenue
Soquel, California 95073

